Algoritma Yazılımı Gerçekleştirme ve Doğrulama Uygulamaları

Mehmet Kadir ASLAN[1] Gözde ÇEVİK ASLAN[2] Emin KÖKSAL[3] Mustafa CANAY[4]

[1, 2, 3, 4] Radar, Elektronik Harp ve İstihbarat Sistemleri (REHİS) Grubu, ASELSAN A.Ş. , Ankara
[1]e-posta: kaslan@aselsan.com.tr [2]e-posta: gcevik@aselsan.com.tr
[3]e-posta: ekoksal@aselsan.com.tr [4]e-posta: mcanay@aselsan.com.tr

Özetçe

Algoritma yazılımları, gömülü yazılımların performansı ve doğruluğu açısından kritik modülleridir. Bu modüllerin geliştirilmesinde ve doğrulanmasında izlenecek yöntemler bu yazılımların ve bu yazılımları kullanan sistemlerin performansını önemli ölçüde etkiler.

Algoritma yazılımlarının mümkün olduğunca bağımsız yazılım modülleri olarak geliştirilmesi, bağımsız olarak test edilmesi ve diğer yazılımlara dışarıdan entegre edilmesinin, algoritma yazılımlarının doğruluğunu ve güvenilirliğini arttırdığı görülmüştür.

Uygulanan yöntem ile hem algoritma yazılım modüllerinin, hem de sistemin diğer modüllerinin geliştirilmesi ve test edilmesi hızlandırılmıştır. Algoritma yazılımlarının modülerliği sayesinde, daha uygun bir algoritma yazılımı geliştirilmesi durumunda, var olan algoritma ile hızlıca değiştirilerek önemli performans iyileştirmeleri sağlanmaktadır.

Bu bildiride algoritma yazılımlarının bağımsız modüller olarak geliştirilmesi ve doğrulama çalışmaları sırasında izlenen yöntemler ve faydaları anlatılmaktadır.

1. Giriş

Algoritma, bir görevi yerine getirmek için uygulanması gereken sonlu sayıdaki adımlar kümesi olarak tanımlanabilir. Her ne kadar adım sayısı sonlu da olsa, büyük ve karmaşık algoritmalar için bu sayılar binleri hatta onbinleri bulabilir. Algoritmaların bu kadar karmaşıklaşması, bu algoritmaları geliştirmek için sistematik yaklaşımları zorunlu kılmıştır. Aynı şekilde bu ölçekteki algoritmaların gerçekleştirilmesi de bazı sistematik yaklaşımlar gerektirmektedir.

Bir algoritmayı gerçekleştirmeye başlamadan önce yapılması gereken ilk iş algoritmanın geliştirilmesidir. Algoritma geliştirme çalışmaları, ilgili algoritmanın, kendinden beklenen görevleri yerine getirebilmesi için izlemesi gereken yolları ve alt adımları belirleme işidir. Algoritma geliştirme çalışmaları kapsamında algoritmanın tasarımı yapılır, ardından da tasarlanan algoritma simülasyon ortamında kodlanarak, çeşitli test verileri ile tasarım doğrulanır.

Geliştirme aşaması tamamlanmış olan algoritmayı kullanarak bir görevi yerine getirecek bir Yazılım Konfigürasyon Birimi (YKB), bu algoritmayı gerçekleştirmeli ve uygun şekilde kullanmalıdır. Ancak bir YKB'nin, bu algoritmanın görevini yerine getirmesinin yanında yapması gereken başka işler de olacaktır. Hatta büyük sistemlerde bu YKB'nin çalıştırması gereken başka algoritmalar da olacaktır. Bu sebeple, bir YKB için, kullanacağı algoritmanın gerçekleştirilme yöntemi önem kazanmaktadır.

Algoritma gerçekleştirme yöntemleri arasında ilk akla gelen, algoritmayı yazılımın bir parçası olarak gerçekleştirmeye çalışmaktır. Ancak bu durumda, algoritma yazılımı kodlamalarının, YKB kapsamında geliştirilen diğer kodlarla koordineli şeklide yürütülmesi, sorunsuz çalışan bir YKB'ye sahip olmak için gereklidir.

Aksi yönden, YKB'nin başka bölümlerinde çıkan problemlerden veya başka bir algoritmadaki problemlerden dolayı, gerçekleştirilmekte olan algoritmanın testlerinin devam edememesi istenen bir durum değildir. Aynı zamanda, algoritmayı kullanacak YKB'nin, algoritmanın geliştirilme süreci boyunca beklememesi, YKB'nin algoritma harici kısımlarının kodlama ve testlerinin devam etmesi arzu edilir.

Ayrıca aynı algoritmanın birden fazla YKB tarafından kullanılması gerekebilir. Bu durum, algoritma kodunun kendinden bekleneni, YKB'nin diğer kısımlarından bağımsız olarak, yerine getiren bir kod olması gereğini doğurur.

Sayılan ihtiyaçlardan dolayı, uygulanacak en doğru yöntemin, "algoritmanın bağımsız modüller olarak gerçekleştirilmesi ve kendisini kullanacak YKB'lere dışarıdan entegre edilmesi" olduğu açıktır.

Bu bildiride bahsedilecek algoritma yazılımları, ASELSAN REHİS bünyesindeki Elektronik Harp projelerinde kullanılmak üzere geliştirilen ve gerçekleştirilen yazılımlardır. Bu algoritma yazılımları, kendinden beklenen göreve özgü; ama projeden bağımsız olarak, birden fazla projenin ihtiyacını karşılayabilecek şekilde geliştirilen matematiksel yazılım parçalarıdır.

Bu makalede, algoritmaların gerçekleştirilmesi sırasında izlenen yöntemler ve bu yöntemlerin faydaları anlatılmaktadır. 2. bölümde algoritma geliştirme, 3. bölümde algoritma yazılımı gerçekleştirme, 4. bölümde algoritma yazılımını doğrulama işlemleri ve 5. bölümde ise algoritma yazılımlarının projelere nasıl entegre edildiği anlatılmıştır.

2. Algoritma Geliştirme

"Algoritma geliştirme" kavramı, bir görevi yerine getirmek için uygulanacak adımları belirleme işlemlerini ifade eder.

Algoritma geliştirme, algoritma gerçekleştirme çalışmalarından bağımsız bir süreç gibi görünmesine rağmen, bu iki çalışma birbirini besler şekilde yürütülmelidir. Çoğunlukla bu işlemlerden biri sırasında bulunan sonuçlar diğeri için bir geri besleme niteliğindedir. Bu nedenle, bu makalede algoritma geliştirme çalışmalarına da kısaca değinilmesi uygun bulunmuştur.

Algoritma geliştirme çalışmaları kapsamında ASELSAN'da benimsenen yöntem, aşağıdaki adımları izlemeyi gerektirir.

Algoritmanın Görevini Tanımlama: Öncelikle bu algoritmanın yapacağı işler net bir şekilde tanımlanır. Algoritmanın girdileri, algoritmadan beklenen çıktılar belirlenir.

Algoritmayı Alt Adımlara Bölme: Algoritma uygun şekilde alt adımlara bölünür. Bu alt adımların, kendi içinde bir bütünlük içermesi gereği aranır.

Alt Adımları Tasarlama: Alt adımların her biri ayrı ayrı tasarlanır. Bu sırada, alt adımların birbirleriyle ilişkileri de irdelenir ve arayüzleri belirlenir.

Algoritmayı Test Etmek İçin Gereken Verileri Belirleme: Algoritmayı test etmek için gereken veriler elde edilir. Gerçek test verileri mevcut ise bunlar kullanılır. Ancak çoğunlukla bu aşamada gerçek veriler mevcut değildir ve dolayısıyla test verileri benzetimle üretilir. Benzetimlerin gerçeğe mümkün olduğunca yakın olması önemlidir.

Test Çıktılarını Belirleme: Algoritmanın sonuçlarının nerelerde kontrol edileceği belirlenir. Sonuçların kontrol edileceği yerler ve sayıları geliştirilen algoritmaya ve bu algoritmanın kritikliğine bağlı olarak değişiklik göstermektedir.

Alt Adımları Geliştirme ve Benzetim: Algoritma adımları simülasyon ortamında gerçekleştirilir ve testleri yapılır. Test sonuçlarında algoritmanın beklenen davranışı sergilediği görülmelidir. İstenen sonuçlar elde edilmezse önceki adımlara dönülerek hatalar giderilir.

Algoritmanın Sonuçlarını Kaydetme: Algoritmaların ilgili test verisi için verdiği sonuçlar kaydedilir. Bu sırada tüm aşamalardaki sonuçlar kaydedilmelidir. Bu sonuçlar daha sonra, gerçekleştirilen yazılımın doğrulama testleri için referans olacaktır.

3. Algoritma Yazılımını Gerçekleştirme

Geliştirme çalışmaları tamamlanmış bir algoritma için algoritma gerçekleştirme çalışmaları başlatılır.

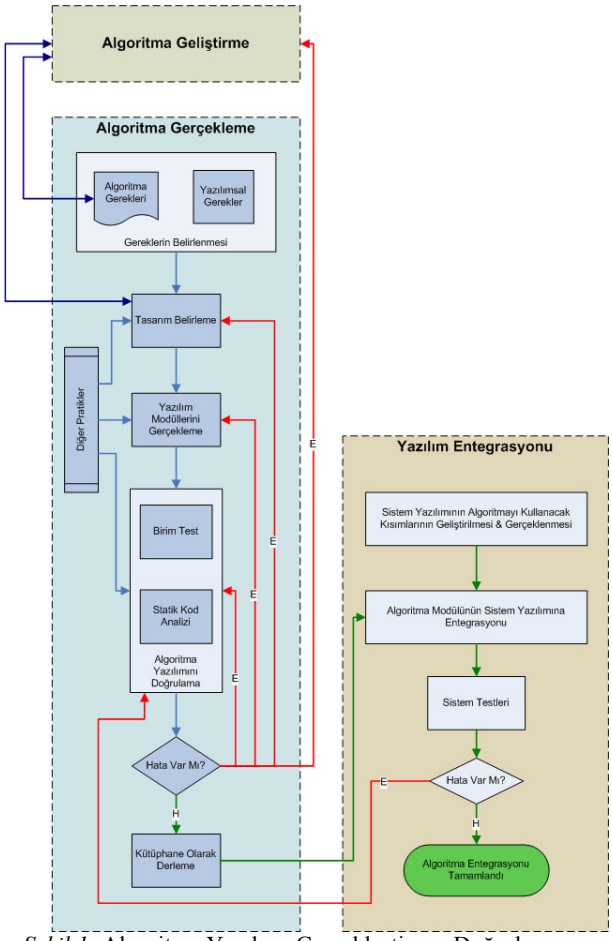
"Algoritma yazılımı gerçekleştirme" kavramı, algoritma geliştirme kapsamında belirlenen adımları kullanarak ilgili görevi yerine getirecek yazılımı üretme işlemlerini ifade eder.

Algoritma gerçekleştirme çalışmalarında, amaç sadece algoritmanın kendinden beklenen gerekleri yerine getirecek bir yazılım üretmek değildir. Bu algoritmanın gerçek zamanlı sistemlerin performans kriterlerini sağlaması, sistemlerin güvenlik kısıtları ile uyumlu olması ve farklı YKB'lerin ihtiyaçlarına göre uyumlandırılabilir olması da algoritma gerçekleştirme aşamasının önemli girdileridir.

Bu makalede önerilen yöntemde, algoritma yazılımı gerçekleştirme sırasında üzerinden geçilen aşamalar *Şekil 1*'de tariflenmektedir.

Şekil 1'de verilen algoritma yazılımı gerçekleştirme aşamaları aşağıda detaylı olarak anlatılmaktadır.

Algoritma Kaynaklı Gerekleri Belirleme: Gerçekleştirilecek algoritma incelenerek, algoritma kaynaklı gerekler belirlenir. Yazılımın temel işi algoritmaya uygun şekilde çalışmak olacağı için, gereklerinin büyük bir bölümü algoritma kaynaklı olmaktadır.



Şekil 1: Algoritma Yazılımı Gerçekleştirme, Doğrulama ve Yazılım Entegrasyonu Aşamaları.

Yazılımsal Gerekleri Belirleme: Algoritma yazılımının, entegre edileceği YKB'lere uygun çalışması için yapması gerekenler belirlenir. Bunlar çoğunlukla girdi ve çıktı veri yapısının nasıl olacağını ve YKB'lere entegrasyonunun nasıl yapılacağını adresleyen gereklerdir. Bu aşamada yazılımın dış arayüzlerinin belirlenmesi ve sabitlenmesi önemlidir. Dış arayüzlerin sonradan değiştirilmeye çalışılmasının maliyeti yüksek olacağından arayüz belirleme işlemleri kodlamadan önce yapılmaktadır.

Algoritma ve Yazılım İhtiyaçlarına Uygun Tasarımı Belirleme: Belirlenen gereklere uygun şekilde yazılımın alt adımlara bölünmesi ve alt adımların tasarlanması yapılır. Her alt adım için girdi, çıktı ve yapacağı işlem belirlenir. Burada uygulanan "alt adımlara bölme" işlemi, algoritma geliştirme aşamasında uygulanan bölme ile aynı veya daha detaylı olabilmektedir. Burada önemli olan, kendi içinde bütünlüğü olan parçalar elde etmektir.

Algoritmik ve Yazılımsal Tasarıma Göre Yazılım Modüllerini Gerçekleştirme: Belirlenen yazılım parçaları, yazılımın çalışacağı ortama uygun bir dille geliştirilir. Algoritmanın karakterine uygunsa, daha genel bir çalışma ortamına uygun şekilde geliştirilip asıl çalışma ortamına göre özelleştirilir. Bu şekilde bir yaklaşım tekrar kullanılabilirliği arttırmaktadır.

Yukarıda bahsedilen adımlar, algoritma gerçekleştirme sürecinde üzerinden geçtiğimiz olmazsa olmaz aşamalardır.

Bu aşamaların dışında, ASELSAN'daki önceki projelerde edinilen deneyimlere dayanarak, algoritma gerçekleştirmeleri sırasında gerekli gördüğümüz pratikler aşağıda tariflenmektedir.

Bağımsız Hafıza Alanı Açma ve Bırakma Arayüzlerine Sahip Olma: Algoritma yazılımları, ilgilendikleri veri miktarının çokluğu ve yoğun işlem yükleri nedeniyle, fazla miktarda belleğe ihtiyaç duyan yazılımlardır. Ancak yazılımın çalışması sırasında dinamik olarak hafıza alanı açıp bırakmak, süre olarak maliyetli bir işlem olduğu için; bu yazılımların kendisine gerekebilecek tüm veriyi çalışma başlangıcında açması tercih edilmektedir. Bunun da ilerisinde, bizim uygulamamızda veri alanı açma işlemi ve açılan alanları bırakma işlemleri dış arayüze taşınmıştır. Böylelikle hafıza alanı açma işleminin sorumluluğu, algoritma modülünü çağıran yazılıma bırakılmıştır. Ayrıca açılacak veri alanlarının boyutu da ayarlanabilir hale getirilmiştir. Bu sayede, algoritmanın kullanılacağı sistemin veri işleme gereklerine göre uyumlandırılabilecek esnek bir yapı kurulmuştur. Sağlanan bu esneklik sayesinde, algoritma modülünü çağıracak YKB, kendi akışına uygun olan noktada veri alanlarını yaratabilmekte, algoritmayı sürekli çalışır halde kullanabilmekte ya da gerektiği zaman algoritmayı çalıştırıp işlem bitince sonlandırabilmekte ve istediği zaman algoritma için ayırdığı veri alanlarını serbest bırakabilmektedir.

Konfigure Edilebilir Olma: Algoritma yazılımlarının çalışmaları sırasında üretecekleri sonuçlar ve bu sonuçları üretme performansları bazı parametrelere bağlıdır. Bu parametrelerin ayarlanabilir olması, farklı proje ihtiyaçlarına cevap verme konusunda algoritmaya esneklik kazandırmaktadır. Esnekliği daha da artıran bir yapı olması için, bu ayarların, algoritmayı çağıran yazılım tarafından, sistem çalışması sırasında yapılabilmesi sağlanmıştır. Ayrıca algoritma yazılımı aynı anda birden fazla örnek çalıştırabilme yeteneğine de sahiptir. Bu sayede, algoritmayı kullanan YKB, birden fazla algoritma örneğini, her örnek farklı konfigürasyonlarda çalışacak şekilde kullanmakta; bunlara paralel işlem yaptırarak, sistem performansı açısından iyileştirmeler sağlanmaktadır.

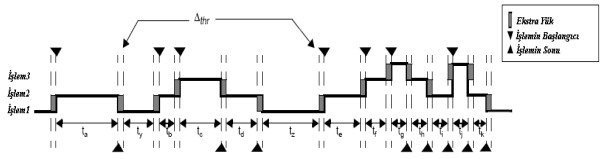
Birden Fazla Projede Kullanılabilir Olma: Önerilen yöntem sayesinde, aynı algoritmaya ihtiyaç duyan birden fazla projede aynı algoritma yazılımı kullanılarak, geliştirme süreleri önemli ölçüde kısaltılmaktadır. Yazılımın konfigure edilebilir olması ve dışarıdan entegre edilebilir olması bu durumu daha da kolaylaştırmaktadır.

Kütüphane Olarak Entegrasyon: Algoritma yazılım modülünü YKB'ye entegre etmek için, "kaynak kod olarak entegrasyon" ya da "kütüphane olarak entegrasyon" seçenekleri mevcuttur. ASELSAN Elektronik Harp projelerindeki uygulamada "kütüphane olarak entegrasyon" tercih edilmiştir. Bu durum, kaynak kod olarak entegrasyona göre hem derleme zamanını kısaltmakta, hem de tekrar derlemekten kaynaklanabilecek hataları engellemektedir. Kütüphane olarak entegrasyon yönteminde, algoritma yazılımının versiyonu kütüphane oluşturulurken sabitlenir ve versiyon kontrolü yazılımı gerçekleştiren kişinin sorumluluğundadır. Kaynak kod olarak entegrasyonun aksine, YKB derlendikçe algoritma yazılımının versiyonu ilerlemez. Bu seçim sayesinde, algoritma modülünde bir hata tespit edilmesi durumunda, modülün aynı versiyonu, test ortamında tekrar çalıştırılarak kendi başına test edilebilmektedir. Bu durum *Şekil 1*'de de ifade edilmektedir.

Girdi ve Çıktılarını Yazdırma Arayüzü: Algoritma yazılımlarının girdileri, çıkaracakları sonucu belirleyen temel unsurdur. Bu sebeple, üretilen sonucun doğruluğundan herhangi bir şüphe duyulduğu anda verilen girdi ve üretilen çıktıları kaydetmek önemli getiriler sağlamaktadır. Algoritma yazılımlarınca kaydedilen bu veriler test ortamında tekrar çalıştırılarak hata durumu yeniden yaratılır. Problemler bu şekilde tespit edilerek giderilmeye çalışılır. Bu veriler kullanılarak yapılan testler, sistemden bağımsız olarak test ortamında yapılır. Bu sayede, sistemin normal çalışması sırasında oluşan problemler tekrar yaratılıp çözülebilmektedir.

Farklı Noktalardan Giriş Yapılabilme: Algoritma yazılımları genel olarak tek bir görevi yerine getirmek üzere gerçekleştirilirler. Ancak bazı durumlarda algoritma yazılımını farklı noktalardan başlatarak çalıştırabilmek önemli olmaktadır. Böyle bir durum yazılımın testi için kolaylık sağlar. Bu sayede problem olan yer belirlenip problem daha hızlı çözülmektedir. Ayrıca bu durum konfigure edilebilirliği de arttırır. Bir YKB, algoritmayı en baştaki adımıyla başlatırken; başka bir projenin YKB'si, farklı ihtiyaçlarından ya da donanımsal özelliklerinden dolayı, algoritmanın bazı adımlarına ihtiyaç duymayarak, algoritmayı daha sonraki bir adımdan başlatabilmektedir.

Sürekli Takip Etme ve Zamanlama Verisi Biriktirme: Algoritma yazılımlarının performansı, sistemlerin performansını belirleyeceği için büyük önem taşımaktadır. Bu yazılımların performansının sürekli ölçülmesi ve mümkün olduğunca iyileştirilmesi gereklidir [1]. Ayrıca yazılımın hangi dallardan geçtiğini, bu dallarda hangi verileri kullandığını takip edebilmek de önemlidir. Bu amaçlarla yazılımın hangi noktalardan, hangi zamanlarda geçtiğini kaydeden başka bir modül gerçekleştirilmiş ve algoritma yazılımlarına entegre edilmiştir. Bu modül, algoritmanın kritik noktalarında hata takibi ve performans ölçümü için kullanılabilecek bir miktar veriyi de kaydetmektedir. Bu modülün getirdiği ekstra yükün, algoritmanın çalışma süresinin yanında ihmal edilebilir derecede küçük olması gerekir [1]. Yapılan ölçümlerde bu modülden kaynaklanan ekstra yükün 1us'den az olduğu görülmüştür. Veri toplama modülünün, algoritma zamanlarına getirdiği yük *Şekil 2*'de oransal olarak ifade edilmektedir.

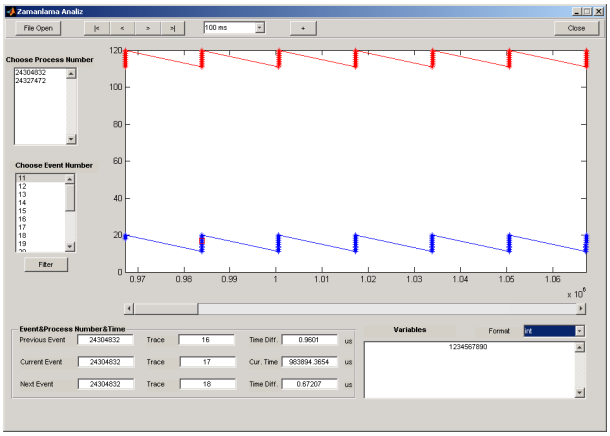


Şekil 2: Veri toplama modülünün algoritma çalışma zamanlarına getirdiği ekstra yükün oransal gösterimi.

Veri kayıtlarına ek olarak, algoritmanın zamanlama performansını ölçmek için, yazılımın içinde çeşitli noktalara "Takip Noktaları" konulmuş, bu noktalardaki zaman bilgilerinin aynı modül ile toplanması sağlanmıştır.

Kritik noktalardan toplanan veriler ve zaman bilgileri çalışma sırasında biriktirilmekte ve algoritma çalışması sonlandığında dışarı aktarılmaktadır. Daha sonra bu bilgiler harici olarak

başka bir yazılım ile analiz edilmektedir. *Şekil 3*'te analiz aracının örnek bir görüntüsü verilmektedir. Bu analiz sonuçlarında yazılımın performansı ölçülmekte, bir hata varsa nereden kaynaklandığı tespit edilebilmektedir.



Şekil 3: Zamanlama verileri analiz aracı.

Bahsi geçen modül, algoritma yazılımı teslim edildikten sonra da yazılımda kalacaktır. Bu sayede, görev sırasında da kritik veriler toplanmaya devam edilecek ve gerektiğinde incelenebilecektir.

4. Algoritma Yazılımını Test Etme

Gerçekleştirilen algoritma yazılımlarını doğrulama aşamasına gelindiğinde; algoritma yazılımının, algoritma geliştirme aşamasında belirlenen test verileri için algoritma simülasyonunun verdiği sonuçları verip vermediği kontrol edilir.

"Birim Test" olarak adlandırılan bu test yönteminde algoritmanın tüm gereklerinin hatasız şekilde gerçekleştirildiği doğrulanır. Algoritma yazılımının birim testi, önce hazır test aracıyla sonra da ASELSAN'da kurduğumuz otomatik test altyapımızla olmak üzere iki aşamada yapılmaktadır.

Öncelikle, her iki aşama için de geçerli olmak üzere, yazılım için kritik olan çıktı noktaları belirlenir.

Hazır test aracıyla yapılan testlerde, kontrollü test verileri ve seçilen kritik noktaların bu verilerle üretmesi beklenen sonuçlar test aracına girilir. Test aracı, yazılımı kontrollü test verilerini kullanarak koşturur. Çıkan sonuçların istenen sonuçlar olup olmadığını kontrol eder. Bunun yanında, birim testin bir diğer amacı da algoritma yazılımının akış diyagramını görmektir. Test verilerinin kod içerisindeki bütün olası yolları kapsamadığı durumlar olabilmektedir. Bu diyagram incelenerek hangi test verilerinin yazılımda hangi yolu takip ettiği, kapsanmayan (dolayısıyla test edilememiş) yolların olup olmadığı tespit edilir. Bu durumda, o yolların da kullanılmasını sağlayan yeni test verileri üretilebilmekte, ya da yazılımda hiç koşturulamayan gereksiz kod parçaları olduğuna karar verilip bunlar ayıklanabilmektedir.

Hazır test aracı, test ettiği kodun senaryosal gerekleri konusunda bilgi sahibi değildir. Dolayısıyla, hazır test aracı ile yapılan birim testlerin kapsamı, bu aracın test yetenekleri ile sınırlıdır. Bundan dolayı, hazır test aracından ayrı olarak, algoritma gerçekleştiricisi tarafından bir birim test altyapısı oluşturulmuştur [2].

Hazır test aracı kullanılmadan yapılacak birim testler kapsamında kontrol edilecek veri miktarı ve kritik veri noktası çok fazla olabilmektedir. Ayrıca tespit edilen bir hata sonucunda kodda düzeltme yapıldıysa, aynı test verileriyle kodun tekrar çalıştırılıp sonuçların eskileriyle karşılaştırılması gerekmektedir. Buna ek olarak, kodda yapılan düzeltmenin algoritmanın başka bölümlerinde yeni bir hataya yol açmadığının kontrolü önemlidir.

Bu sebeple testlerin otomatik olarak yapılması ve sonuçların kaydedilmesi için bir altyapı kurulmuştur. Benzer bir altyapı algoritma geliştirme aşamasında da kurulmuştur. Bu sayede hem algoritma simülasyonu hem de algoritma yazılımı tekrar tekrar çalıştırılıp sonuçlar otomatik olarak kaydedilebilmektedir. İki tarafta kaydedilen veriler başka bir yazılım tarafından otomatik olarak karşılaştırılır. Bu yazılım farklı olan sonuçları gösterir. Farklılığın sebebi araştırılır ve uygun şekilde çözülür.

Algoritma yazılımlarının testi için, "Birim Test"in dışında bir de statik kod analizleri uygulanmaktadır. "Statik Kod Analizi" olarak adlandırılan bu testlerde, seçilen kodlama standardına göre kodda bulunan uygunsuzluklar ve hataya açık noktalar tespit edilerek düzeltilir.

Algoritma yazılımının statik kod analizi için hazır alınan bir test aracı kullanılmaktadır. Bu test aracı yardımıyla, seçilen kodlama standardına göre algoritma kodlarının statik analiz sonucu elde edilir. Statik kod analizi sonuçları, algoritma yazılımının gerçekleştirilmesi sırasında yapılan standarda aykırı kodlamaları içerir. Buna örnek olarak "for", "while", "break", "return" gibi koda özel anahtar kelimelerin kullanımı, matematiksel işlemler, hafıza alanı ile ilgili işlemler verilebilir. Standartlar, bu ifadelerin hatayı en çok engelleyecek şekilde nasıl kullanılması gerektiğini belirlemiştir. Bu standartlara aykırı durumlar analiz sonucunda görülüp düzeltilir. Böylelikle gerçek çalışmada oluşabilecek hata riski en aza indirgenmekle birlikte, standartlara uygun biçimde kod yazılarak anlaşılabilirlik ve kod kalitesi arttırılmış olur. [3]

Anlatılan birim test ve statik test yöntemleri, birbirlerine alternatif olarak değil, birbirlerini tamamlayıcı şekilde düşünülmelidir (Bkz. *Şekil 1*). Statik kod analizleri sayesinde, yapılan hatalar daha kodlama aşamasındayken düzeltilmektedir. Böylece erken dönemde kod iyileştirmelerine olanak sağlanmaktadır. Dinamik kod analizi olarak düşünülen birim testler ile de algoritma yazılımlarının gerçek verilerle testleri yapılmakta, yazılımda çalışma esnasında görülebilecek hatalar gerçek sisteme entegre edilmeden tespit edilip düzeltilebilmektedir.

Bu biçimde bir test yöntemi ile projelere tüm testleri tamamlanmış bir algoritma yazılımı verilmektedir. Bu sayede, bu yazılımı kullanan projelerin tekrar test ihtiyacı minimuma indirilmiştir [4].

5. Algoritma Yazılımının Projelere Entegrasyonu

Gerek analizleri yapılmış, statik kod analizleri ve birim testlerle bağımsız bir modül olarak doğrulanmış olan algoritma yazılımı, bu noktadan sonra sistem yazılımlarına entegre edilmeye hazırdır.

Algoritma yazılımları sistem yazılımlarından bağımsız geliştirildiği için, bu süre zarfında algoritmayı kullanacak

sistem yazılımı da başka bir kodlayıcı tarafından paralelden gerçekleştirilebilmektedir.

Sistem yazılımı, kütüphane olarak derlenmiş olan algoritma yazılımını kendi projesine ekler. Algoritma yazılımının dış dünyaya açılım noktaları olan "C/C++ Başlık" dosyaları da projeye dahil edilir. Bu dosyalardaki değişime açık parametreler, proje ihtiyaçlarına göre ayarlanarak, algoritma projeye uygun konfigürasyona getirilmiş olur.

Algoritmanın ihtiyaç duyduğu veri alanları sistem yazılımı tarafından açılarak algoritmaya bildirilir. Bu şekilde, hafıza taşmalarının önüne geçmek için gerekli kontrol, sistem yazılımının eline verilmiştir.

Bu noktadan itibaren, sistem yazılımını kodlayan kişi algoritmayı bir "kara kutu" gibi algılar. Sistem yazılımı sadece algoritmaya gerekli girdileri doğru şekilde aktarmaktan ve sonuçları doğru formatta toplamaktan sorumludur. İlgili girdi verileri için elde edilen sonuç verilerinin doğru olup olmadığı sistem yazılımının kapsamı dışındadır. Sistem yazılımcısı, algoritmanın doğru çalışıp çalışmadığından bağımsız olarak, yalnız kendi gereklerini test eder.

Sistem yazılımının algoritma modülüne ihtiyaç duyduğu zamanda algoritma henüz hazır değilse, gerçek algoritma yerine aynı amaç için eskiden geliştirilmiş bir algoritma ya da ticari hazır yazılımlar kullanılır. Beklenen algoritma yazılımı hazır olduğunda, eski yazılımın bir kara kutu halinde sökülüp yerine yeni algoritmanın konması gerekmektedir. Algoritma yazılımının "kütüphane olarak entegre edilecek bir modül" olarak geliştirilmiş olması, bu işlemin oldukça kolaylaşmasını sağlamıştır.

Ayrıca gerçek sistemde gerçek verilerle çalışma sırasında algoritmada daha önce oluşmamış bir hata görülürse (Bkz. *Şekil 1*), sistem yazılımı algoritmanın mevcut halini ya da bir önceki versiyonunu kullanmaya devam ederken, paralelden algoritma kodlayıcı kendi yazılımındaki hatayı bulur ve düzeltir. Yeni algoritma versiyonunun sistem yazılımına entegrasyonu, yazılıma dahil edilen kütüphanenin değiştirilmesinden ibaret olmaktadır.

6. Tartışma

Uygulanan bu yöntem birçok problemi çözüme kavuşturmuştur. İlk bakışta, bu yapının detaylarla fazla uğraştığı ve geliştirme sürecini uzattığı izlenimi doğabilir. Ancak unutulmamalıdır ki, bu yöntem kullanılmadığı durumda entegrasyon ve arazi testleri sırasında aynı problemlerle uğraşmak gerekecektir.

Algoritmanın, yazılımın diğer kısımlarından ayrı tutularak geliştirildiği bu yaklaşımla, algoritmadaki versiyon değişikliklerinin yazılıma entegrasyonu oldukça kolaylaşmıştır [4].

Bu yöntem sayesinde; algoritmanın henüz yeni geliştirilmekte olduğu aşamada, eldeki mevcut eski algoritma geçici olarak kullanılmış, yeni algoritma tamamlandığında eskisini bir kara kutu gibi çıkarıp yerine yenisini koymak mümkün olmuştur. Bu durum, yazılımın bu algoritmaya ihtiyaç duyan diğer kısımlarının da algoritmayla paralel olarak geliştirilmesine olanak sağlamıştır.

Daha önce geliştirilmiş projelerde uygulanan yöntemlerden edinilen tecrübeler, böyle kontrollü şekilde geliştirmeyi zorunlu kılmıştır [5].

7. Kazanımlar

Bu yöntemin faydaları aşağıdaki gibi sıralanabilir.

- Algoritma geliştirmeyi ve yazılım geliştirmeyi hızlandırır.
- Algoritma simülasyonu ve algoritma yazılımı bağımsız test edilir.
- Gerek algoritma yazılımı açısından, gerekse sistem YKB'si açısından test yapmayı hızlandırır.
- Sistem YKB'si ve algoritma yazılımı birbirlerinden bağımsız ve arayüzleri de belirli olduğu için, hata kaynağı doğru tespit edilir.
- Hatalar, gerçek sisteme entegrasyondan çok önce tespit edilerek düzeltilir.
- Algoritma yazılımının sistem YKB'sine entegrasyonunu kolaylaştırır.
- Algoritma yazılımında değişiklik yapmak ve bunun sistem YKB'sine yansımasını sağlamak kolaydır.
- Algoritma yazılımını başka bir algoritma yazılımı ile değiştirmek kolaydır.

8. Sonuçlar

Bu bildiride gömülü yazılımlarda kullanılan algoritmaları yazılımdan bağımsız bir modül olarak geliştirmek amacıyla yapılan çalışmalar anlatılmıştır. Algoritmaların geliştirilmesi ve algoritma yazılımlarının gerçekleştirilmesinde uygulanan yöntem detaylı olarak incelenmiştir.

Algoritmaların geliştirilmesinde uygulanan bu yöntemin o algoritmayı kullanacak sistem yazılımlarına entegrasyon sırasında sağladığı faydalara değinilmiştir.

Algoritma ile sistem yazılımının iç içe geliştirildiği yönteme göre elde edilen kazanımlar oldukça önemlidir. Bu yüzden bu uygulamanın ileriye dönük olarak tüm algoritma gerçekleştirme çalışmalarında kullanılması değerlendirilmiştir.

9. Teşekkür

Bu makalenin hazırlanması sırasında desteğini esirgemeyen ve algoritma kodlamaları süresince bu uygulamanın geliştirilmesine katkıda bulunan tüm iş arkadaşlarımıza teşekkürlerimizi sunarız.

10. Kaynakça

[1] David B. Stewart – InHand Electronics, "Measuring Execution Time and Real-Time Performance", *Embedded Systems Conference Boston, 2008*

[2] Ark Khasin - Macro Expressions, "C/C++ Code Unit Testing on a Shoestring", *Embedded Systems Conference Boston, 2008*

[3] Dan Saks, "Writing Better C and C++ for Embedded Systems", *Embedded Systems Conference Boston, 2008*

[4] Bedir Tekinerdoğan – University of Twente, "Software Architecture Engineering / Product Line Engineering Lecture Notes", *Aselsan, 2006*

[5] Jack Ganssle, "Learning From Disaster", *Embedded Systems Conference Boston, 2008*